Refine Search

Search Results -

Term	Documents
(40 AND 48).USPT.	36
(L40 AND L48).USPT.	36

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L49

Refine Search | Recall Text | Clear | Interrupt

Search History

DATE: Tuesday, March 30, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT; PLUR=YES; OP=ADJ			
L49	l40 and L48	36	L49
L48	loop and modif$	241470	L48
L47	l40 and L46	0	L47
L46	self loop	184	L46
L45	l40 and L44	0	L45
L44	self instruction	92	L44
L43	l40 and L42	1	L43
L42	atomic	75224	L42
L41	l15 and l40	0	L41
L40	l37 and L38	36	L40
L39	l37 and L38	36	L39
L38	loop and self	63844	L38
L37	l1 and L36	36	L37

h e b b cg b e e ch

L36	l17 and L34	100	L36
L35	l25 and L34	10	L35
L34	exchange instruction	303	L34
L33	l25 and L32	13	L33
L32	match$ near instruction	826	L32
L31	l28 and L30	0	L31
L30	self loop	184	L30
L29	l25 and L28	2	L29
L28	compare near2 exchange	142	L28
L27	l19 and L26	1	L27
L26	l23 and l24 and l12	3	L26
L25	lock$ mechanism	50831	L25
L24	multithread$	1286	L24
L23	call instruction	1754	L23
L22	l20 and L21	4	L22
L21	binary	150428	L21
L20	l18 and L19	11	L20
L19	runtime	4753	L19
L18	l16 and L17	11	L18
L17	lock$	572654	L17
L16	l14 and L15	13	L16
L15	thread safe	184	L15
L14	self modifying	463	L14
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ			
L13	l11 and L12	19	L13
L12	lock$	1351001	L12
L11	l1 and L10	23	L11
L10	thread safe	351	L10
L9	l1 and L8	3	L9
L8	atomic compare	57	L8
DB=USPT; PLUR=YES; OP=ADJ			
L7	l5 and L6	0	L7
L6	table function	2335	L6
L5	(callback function).ab.	20	L5
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=ADJ			
L4	l1 and l2 and L3	5	L4
L3	multithreading	788	L3
L2	lock	1045646	L2
L1	self modifying	650	L1

END OF SEARCH HISTORY

h e b b cg b e e ch

L49: Entry 34 of 36 File: USPT Jun 16, 1998

DOCUMENT-IDENTIFIER: US 5768610 A
TITLE: Lookahead register value generator and a superscalar microprocessor employing same

Drawing Description Text (8):
While the invention is susceptible to various modifications and alternative forms, specific embodiments thereof are shown by way of example in the drawings and will herein be described in detail. It should be understood, however, that the drawings and detailed description thereto are not intended to limit the invention to the particular form disclosed, but on the contrary, the intention is to cover all modifications, equivalents and alternatives falling within the spirit and scope of the present invention as defined by the appended claims.

Detailed Description Text (11):
The bit-encoded execution instructions and immediate data provided at the outputs of decode units 208A-208F are routed directly to respective reservation station units 210A-210F. In one embodiment, each reservation station unit 210A-210F is capable of holding instruction information (i.e., bit encoded execution bits as well as operand values, operand tags and/or immediate data) for up to three pending instructions awaiting issue to the corresponding functional unit. It is noted that for the embodiment of FIG. 1, each decode unit 208A-208F is associated with a dedicated reservation station unit 210A-210F, and that each reservation station unit 210A-210F is similarly associated with a dedicated functional unit 212A-212F. Accordingly, six dedicated "issue positions" are formed by decode units 208, reservation station units 210 and functional units 212. Instructions aligned and dispatched to issue position 0 through decode unit 208A are passed to reservation station unit 210A and subsequently to functional unit 212A for execution. Similarly, instructions aligned and dispatched to decode unit 208B are passed to reservation station unit 210B and into functional unit 212B, and so on. After aligning the instructions to the issue positions, the instructions are scanned for PUSH and POP instructions. A constant value is generated for each instruction indicative of the amount that the ESP register will be modified by PUSH or POP instructions prior to and including that instruction with respect to the ESP register value prior to executing the instruction in the first issue position. Additionally, a special constant encoding is generated for the case of a single POP instruction (with no PUSH instructions) being contained within the set of issue positions. Another special constant encoding is indicative of a single PUSH instruction (with no POP instructions) being contained within the set of issue positions. The special constant encodings are used by decode units 208 as a flag to indicate the above listed conditions. This constant value is conveyed to a decode unit 208 along with the instruction. Suitable scanning mechanisms are well known.

Detailed Description Text (12):
Upon decode of a particular instruction, if a required operand is a register location, register address information is routed to reorder buffer 216 and register file 218 simultaneously. Those of skill in the art will appreciate that the x86 register file includes eight 32 bit real registers (i.e., typically referred to as EAX, EBX, ECX, EDX, EBP, ESI, EDI and ESP). Reorder buffer 216 contains temporary storage locations for results which change the contents of these registers to thereby allow out of order execution. A temporary storage location of reorder

buffer 216 is reserved for each instruction which, upon decode, is determined to modify the contents of one of the real registers. Therefore, at various points during execution of a particular program, reorder buffer 216 may have one or more locations which contain the speculatively executed contents of a given register. If following decode of a given instruction it is determined that reorder buffer 216 has a previous location or locations assigned to a register used as an operand in the given instruction, the reorder buffer 216 forwards to the corresponding reservation station either: 1) the value in the most recently assigned location, or 2) a tag for the most recently assigned location if the value has not yet been produced by the functional unit that will eventually execute the previous instruction. If the reorder buffer has a location reserved for a given register, the operand value (or tag) is provided from reorder buffer 216 rather than from register file 218. If there is no location reserved for a required register in reorder buffer 216, the value is taken directly from register file 218. If the operand corresponds to a memory location, the operand value is provided to the reservation station unit through load/store unit 222.

Detailed Description Text (14):
Reservation station units 210A-210F are provided to temporarily store instruction information to be speculatively executed by the corresponding functional units 212A-212F. As stated previously, each reservation station unit 210A-210F may store instruction information for up to three pending instructions. Each of the six reservation stations 210A-210F contain locations to store bit-encoded execution instructions to be speculatively executed by the corresponding functional unit and the values of operands. If a particular operand is not available, a tag for that operand is provided from reorder buffer 216 and is stored within the corresponding reservation station until the result has been generated (i.e., by completion of the execution of a previous instruction). It is noted that when an instruction is executed by one of the functional units 212A-212F, the result of that instruction is passed directly to any reservation station units 210A-210F that are waiting for that result at the same time the result is passed to update reorder buffer 216 (this technique is commonly referred to as "result forwarding"). Instructions are issued to functional units for execution after the values of any required operand (s) are made available. That is, if an operand associated with a pending instruction within one of the reservation station units 210A-210F has been tagged with a location of a previous result value within reorder buffer 216 which corresponds to an instruction which modifies the required operand, the instruction is not issued to the corresponding functional unit 212 until the operand result for the previous instruction has been obtained. Accordingly, the order in which instructions are executed may not be the same as the order of the original program instruction sequence. Reorder buffer 216 ensures that data coherency is maintained in situations where read-after-write dependencies occur.

Detailed Description Text (23):
Decode unit 208A receives an instruction and an associated constant value from instruction alignment unit 206 on an instruction bus 250. Instruction bus 250 is coupled to an instruction register 251. The output of instruction register 251 is coupled to a decode control unit 253. A segment request bus 254 is also coupled to decode control unit 253, and is coupled to load/store unit 222 (shown in FIG. 1). Decode control unit 253 produces several input values for a multiplexor 256 which include a displacement value, an immediate value, and a constant value. The displacement value is conveyed on a displacement bus 258; the immediate value is conveyed on an immediate bus 259, and the constant value is conveyed on a constant bus 260. Select signals for multiplexor 256 and a multiplexor 257 are generated by decode control unit 253 and are conveyed on select bus 261 and select bus 262, respectively. Multiplexor 257 is configured to select between lookahead register values conveyed on lookahead EBP bus 263, lookahead ESP bus 264, and transfer bus 265. Lookahead EBP bus 263 and lookahead ESP bus 264 are coupled to a lookahead control unit 226 (shown in FIG. 3). Transfer bus 265 is coupled to load/store unit 222 and to reorder buffer 216 (both shown in FIG. 1). The outputs of multiplexors

256 and 257 are coupled to adder circuit 266, the output of which is coupled to a multiplexor 267. Multiplexor 267 is also coupled to lookahead ESP bus 264 and a lookahead ESP-4 bus 268 (which is coupled to lookahead control unit 226). The output of multiplexor 267 is coupled to yet another multiplexor 269 which additionally receives a register tag bus 270 from decode control unit 253. Multiplexors 267 and 269 receive their select signals (not shown) from decode unit 253. An ESP/EBP modification/correction unit 274 is coupled to a decoded instruction register 276 which stores a decoded instruction. ESP/EBP modification/correction unit 274 is further coupled to an output lookahead register bus 273A and a first control bus 275A. The output of instruction address register 271 is conveyed to ESP/EBP modification/correction circuit 274 and then to reservation station 210A on output lookahead register bus 273A. Lookahead address register 271 is also coupled to multiplexor 256 as an input. The decoded instruction stored in decoded instruction register 276 is conveyed on decoded instruction bus 277, which is further coupled to reservation station 210A. Output lookahead register bus 273A is used to convey a lookahead register value. Additionally, output lookahead register bus 273A is used to convey lookahead addresses and register tag data to reservation 210A, as described below.

Detailed Description Text (32):
ESP/EBP modification/correction unit 274 receives the decoded instruction stored in decoded instruction register 276. ESP/EBP modification/correction unit 274 detects if the instruction modifies ESP or EBP in a fashion that is inconsistent with the address stored in lookahead address register 271. If such a modification is detected, then ESP/EBP modification/correction unit 274 attempts to create the modified ESP or EBP value for certain instructions. In one embodiment, these instructions are the move from ESP to EBP, move from EBP to ESP, add immediate data to the ESP or EBP, and subtract immediate data from the ESP or EBP. If these instructions are decoded by decode control unit 253, then decode control unit 253 stalls the pipeline for one clock cycle to allow this calculation to be made. The corrected address is conveyed on output lookahead register bus 273A to lookahead control unit 226 and reservation station 210A. If no correction is detected, the value stored in instruction address register 271 is conveyed on output lookahead register bus 273A. A value is also conveyed on first control bus 275A indicative of which register (ESP, EBP, or no register) is being modified with the value conveyed on output lookahead register bus 273A. A second value is conveyed on first control bus 275A indicative of a move from ESP to EBP or a move from EBP to ESP (or neither). This value is used by lookahead control unit 226 to cause the ESP value to be copied into the EBP value, or the EBP value to ESP value, (or neither), respectively.

Detailed Description Text (33):
It is noted that one embodiment of microprocessor 200 is configured to handle up to three PUSH instructions or a single POP instruction in a single clock cycle. If more than one POP instruction, a combination of PUSH and POP instructions, or more than three PUSH instructions are conveyed to decode units 208, the pipeline stalls until each POP instruction is handled individually and each set of up to three PUSH instructions are handled. Furthermore, an instruction which modifies the ESP or EBP with a value that cannot be speculatively generated by decode units 208 causes the pipeline to stall until the instruction completes. The correct value of the ESP or EBP is then transferred to lookahead control unit 226 from reorder buffer 216 on transfer bus 265. Other embodiments may be configured to handle more or fewer combinations of PUSH and/or POP instructions. It is further noted that when a particular decode unit "stalls the pipeline", instructions which are prior (in program order) to the instruction being decoded by the particular decode unit are permitted to progress to reservation stations 210 and functional units 212. However, instructions subsequent to the instruction being decoded by the particular decode unit are not permitted to progress. Instead, these instructions are held until the condition causing the pipeline stall is alleviated. The instruction being decoded by the particular decode unit typically also is allowed to progress, except

for the case of logical to linear address conversion when the flat memory model is not being used. Additional stall sources may also cause the instructions being decoded not to progress. For example, if reservation stations 210 or load/store unit 222 detect full conditions in their respective buffers, pipeline stalls may occur. In one embodiment, the pipeline is stalled by reorder buffer 216 and reservation stations 210 not storing the stalled instructions and by instruction alignment unit 206 presenting the stalled instructions for decoding in a subsequent clock cycle.

Detailed Description Text (35):
Lookahead control unit 226 receives a plurality of output lookahead register buses 273 from decode units 208, one of which is output register bus 273A from decode unit 208A. Additionally, lookahead control unit 226 receives a plurality of first control buses 275 from decode units 208, one of which is first control bus 275A from decode unit 208A. A constant bus 300 is coupled between lookahead control unit 226 and instruction alignment unit 206 (shown if FIG. 1) which conveys the constant value associated with the final PUSH or final POP instruction within the instructions currently being decoded by decode units 208. A second control bus 301 is coupled between lookahead control unit 226 and reorder buffer 216 (shown in FIG. 4). First control buses 275, constant bus 300, and second control bus 301 are coupled to a next ESP/EBP control unit 302, while output lookahead register buses 273 are coupled to a multiplexor 303. The output of multiplexor 303 is coupled to multiplexors 304, 305, and 306. Also coupled to multiplexors 304, 305, and 306 is transfer bus 265. Transfer bus 265 is used to transfer a new value for the ESP or EBP register for the cases in which an instruction modifies the ESP or EBP register such that the new value cannot be predicted at decode of the instruction. For example, a move from EAX to ESP is not predictable because the value of the EAX is not known at the decode stage. Multiplexor 304 is coupled to multiplexor 310 and receives the output of lookahead EBP register 309 as an additional input, while multiplexor 305 is coupled to multiplexor 311 and multiplexor 306 is coupled to multiplexor 312. Multiplexor 310 is coupled to a real EBP bus 313 from reorder buffer 216 and to lookahead EBP register 307 which is coupled to lookahead EBP bus 263. Similarly, multiplexor 311 is coupled to a real ESP bus 314 from reorder buffer 216 and to lookahead ESP register 308 which is additionally coupled to lookahead ESP bus 264. In an analogous fashion, multiplexor 312 is coupled to a real ESP-4 bus 315 from reorder buffer 216 and to lookahead ESP-4 register 309 which is additionally coupled to lookahead ESP-4 bus 268. Lookahead ESP bus 264 is coupled as an additional input to multiplexor 304 as well as to multiplexors 316 and 317. Multiplexors 316 and 317 also receive lookahead EBP bus 263 as an input. Multiplexor 316 is coupled to adder 318, and multiplexor 317 is coupled to adder 319. Both adder 318 and adder 319 receive additional inputs from next ESP/EBP control unit 302. Adder 318 is coupled as an input to multiplexor 305, and adder 319 is coupled as an input to multiplexor 306. Next ESP/EBP control unit 302 is configured to control the selection of multiplexors 303, 304, 305, 306, 310, 311, and 312.

Detailed Description Text (36):
Lookahead EBP value generation within lookahead control unit 226 will now be explained. Lookahead control unit 226 is configured to generated lookahead EBP values for several possible cases. The first case occurs when the instructions being decoded in decode units 208 do not modify the EBP register, and is detected via values conveyed on first control buses 275. The EBP register output is selected by multiplexor 304 and is routed through multiplexor 310 to lookahead EBP register 307, thereby storing the current lookahead EBP value as the lookahead EBP value for the next clock cycle.

Detailed Description Text (37):
In a second case, one of decode units 208 detects an add or subtract instruction which adds or subtracts immediate data to the EBP value. For this case, the correct value is conveyed on an associated one of output lookahead register buses 273 and a

value indicative of EBP register modification is conveyed on an associated one of first control buses 275. Next ESP/EBP control unit 302 processes the information conveyed on first control buses 275 and causes multiplexor 303 to select the correct value, which is then routed through multiplexors 304 and 310 to lookahead EBP register 307. It is noted that if multiple first or second values conveyed on first control buses 275 are indicative of EBP register modification, the value corresponding to decode unit 208A is given highest priority; the value corresponding to decode unit 208B is given second highest priority; etc. In this manner, the first modification (in program order) of the EBP register is made. Since the associated decode unit has stalled the pipeline (as described above), the other indicated modifications will be decoded again in a subsequent clock cycle.

Detailed Description Text (38):
A third case of update to the lookahead EBP register value is an instruction moving the ESP value to the EBP value. This move instruction is common prior to branching to a subroutine in a computer program. Move instructions of this type are detected by decode units 208 and the second value indicative of the move from ESP to EBP is conveyed on one of first control buses 275. Next ESP/EBP control unit 302 then causes multiplexor 304 to select the lookahead ESP register output, which is routed through multiplexor 310 to lookahead EBP register 307. It is noted that if multiple first or second values conveyed on first control buses 275 are indicative of EBP register modification, the value corresponding to decode unit 208A is given highest priority; the value corresponding to decode unit 208B is given second highest priority; etc. In this manner, the first modification (in program order) of the EBP register is made. Since the associated decode unit has stalled the pipeline (as described above), the other indicated modifications will be decoded again in a subsequent clock cycle.

Detailed Description Text (40):
Lookahead ESP and ESP-4 register value generation is also performed by lookahead control unit 226. As with the lookahead EBP value generation described above, several cases may occur with respect to lookahead ESP and ESP-4 register value generation. A first case of lookahead ESP and ESP-4 register value generation occurs if the instructions being decoded by decode units 208 include a single POP instruction, or include up to three PUSH instructions, or do not modify the ESP register. For this case a pair of constant values are added to the current lookahead ESP value to obtain the lookahead ESP (and ESP-4) values for the subsequent clock cycle. In one embodiment, a constant value which may be used to generate the next lookahead ESP value is generated by instruction alignment unit 206 and conveyed on constant bus 300 to next ESP/EBP control unit 302. Next ESP/EBP control unit 302 is configured to route this constant value to adder 318. Additionally, four is subtracted from the constant value by next ESP/EBP control unit 302 and is conveyed to adder 319. The output of lookahead ESP register 308 is selected by multiplexors 316 and 318. Therefore, the appropriate constants are added to the ESP value by adder circuits 318 and 319, respectively. The new ESP value is routed from adder 318 through multiplexors 305 and 311 to lookahead ESP register 308. The new ESP-4 value is routed through multiplexors 306 and 312 to lookahead ESP-4 register 309.

Detailed Description Text (42):
A second case for the update of the lookahead ESP and ESP-4 values is the move from EBP to ESP. This instruction is detected by decode units 208 and conveyed as an encoding of the aforementioned second value on an associated one of first control buses 275 to next ESP/EBP control unit 302. Next ESP/EBP control unit 302 causes multiplexors 316 and 318 to select the lookahead EBP value and routes a constant of zero to adder 318 and a constant of -4 to adder 319. In this manner, adder 318 produces the current lookahead EBP value and adder 319 produces the current EBP value minus four. The value generated by adder 318 is routed through multiplexors 305 and 311 to lookahead ESP register 308. Similarly, the value generated by adder 319 is routed through multiplexors 306 and 312 to lookahead ESP-4 register 309. It

is noted that if multiple first or second values conveyed on first control buses 275 are indicative of ESP register modification, the value corresponding to decode unit 208A is given highest priority; the value corresponding to decode unit 208B is given second highest priority; etc. In this manner, the first modification (in program order) of the ESP register is made. Since the associated decode unit has stalled the pipeline (as described above), the other indicated modifications will be decoded again in a subsequent clock cycle.

Detailed Description Text (43):
In a third case, the lookahead ESP and ESP-4 are updated via an addition or subtraction of immediate data. For this case, multiplexor 303 selects the corrected value from output lookahead register buses 273. The selected value is routed through multiplexors 305 and 311 to lookahead ESP register 308. In a subsequent clock cycle, the ESP-4 value is updated from the new ESP value through multiplexor 317 and adder 319. It is noted that if multiple first or second values conveyed on first control buses 275 are indicative of ESP register modification, the value corresponding to decode unit 208A is given highest priority; the value corresponding to decode unit 208B is given second highest priority; etc. In this manner, the first modification (in program order) of the ESP register is made. Since the associated decode unit has stalled the pipeline (as described above), the other indicated modifications will be decoded again in a subsequent clock cycle.

Detailed Description Text (49):
It is noted that the above discussion details an embodiment of the lookahead register value generator in which the number of instructions handled in a given cycle is a single POP instruction, or up to 3 PUSH instructions, or a single instruction which modifies the ESP or EBP registers. However, similar circuits could be used to configure a lookahead register value generator that may handle more or less of these various types of instructions. It is further noted that although this technique is described in the context of the x86 architecture, the technique may be used with any processor architecture in which registers are modified by an instruction to values that are predictable prior to the instruction's execution.

Detailed Description Text (72):
Hardware detects and forwards memory calculations that hit in the current entries in the stack relative cache since it is possible for addressing modes outside of stack relative accesses to indirectly point to this same region of memory, and the stack cache is treated as modified memory. Because memory operations are a part of most x86 instructions, load/op/store operations may be converted to single issue operations. Processor 500 does this by allowing a single issue to contain as many as three distinct operations. If memory load and store operations outside of the stack relative cache are detected in decode, the pending operation is held in a reservation station, and the load access and addressing calculation are sent the multi-ported data cache. Upon completion of the load operation the reservation station is allowed to issue to the functional unit. Upon completion of execution, the result is either an x86 register or a pending store.

Detailed Description Text (193):
As stated previously, processor 500 executes fast path instructions directly. Three pre-decode bits are associated with each byte of instruction: a start bit, an end bit, and a functional bit. All the external fetched instructions will be latched into the Icache. Only single-byte prefixes of 0x66 and 0x0F are allowed for fast path instructions. Instructions including a second prefix byte of 0x67 are also allowed, and require one extra decode cycle. All other prefixes require extra cycles in decoding or execution using microcode sequences stored in MROM. With these simple prefixes, the instruction bytes need not be modified. The linear valid bit is used for the whole cache-line of instructions (16 bytes). The replacement procedure is controlled by the L2 unit. Along with each line of instruction, the L2 unit directs the Icache on storing the data and tag. The start and end bits are

sufficient to validate the instruction. In cases of branching to the middle of a line or instructions which wrap around to the next line, the start and end bits must be detected for each instruction or else the instruction must be pre-decoded again. The possible cases are branching to the opcode and skipping the prefix (pruning of the instruction) and replacing part of the instruction in the Icache. The instructions must first be passed through pre-fetch buffers before being sent to the ICPRED. The ICPRED has only one input from the IB(127:0) for both the pre-fetched or cached instructions. The pre-decode information is written into the ICPDAT as the whole line is decoded.

Detailed Description Text (196):
The ICSTORE in processor 500 does not include the pre-decode data. The ICSTORE contains 32K bytes of instructions organized as 8 sets of 128 rows by 256 columns. Each of the sets consist of two bytes of instructions. The 8-way associative multiplexing from the 8 TAG-HITs is performed before the data is routed to the ICALIGN block. With this arrangement, the input/output to each set is 16-bit buses. The multiplexing information regarding which byte is to be directed to which decode unit should also be decoded; this topic will be discussed in more detail in the ICALIGN section. For optimal performance, the layout of the column should be 64 RAM cells, precharge, 64 RAM cells, write buffer, and senseamp. The row decoder should be in the middle of the array to drive 128 columns each way, and the precharge and the row decoder should cross in the middle of the array. The self-time column is used to generate internal clock signals for each set of the array. The precharge is gated by the ICLK signal. The instruction is valid by the end of ICLK, the data multiplexed by the TAGHIT signals should be gated by ICLK to be valid for the second ICLK. The two-entry pre-fetch buffers are implemented inside the array with data input from either entry. The output IB bus is driven by either the array or the pre-fetch buffer.

Detailed Description Text (219):
The predecode bits are sent along with the instructions to the decode units. If a part of the line cannot be dispatched to the decode units, no start-byte is sent for that part of the line. The IBDx buses can be pseudo-dynamic buses with precharge using the self-time clock of the array. If the first byte of the decode unit does not have a start-byte, the decode unit passes a NOOP to the functional unit.

Detailed Description Text (221):
The ICPDAT contains 32K of 3-bit pre-decode data organized as 8 sets of 64 rows by 192 columns. Each of the sets consists of two 3-bit pre-decode data. The pre-decode data is decoded into byte-shifting information which is used by the ICALIGN block. The 8-way associative multiplexing from the 8 TAGHITs is performed before the byte-shifting data is routed to the ICALIGN block. In order for the instructions to get to the Idecode in middle of the second ICLK, the decode logic for the byte-shifting should be less than seven gates. Because of this byte-shifting logic, the array for ICPDAT is 64 rows instead of 128 rows for the ICSTORE array. For optimal performance, the layout of the column is 32 RAM cells, precharge, 32 RAM cells, write buffer and senseamp. The row decoder should be in the middle of the array to drive 96 column each way, and the precharge and the row decoder should cross in the middle of the array. The self-time column is used to generate internal clock signals for each set of the array. The precharge is gated by the ICLK signal. The byte-shifting data multiplexed by the TAGHIT should be gated by ICLK to be valid for the second ICLK. The output of the array should include logic to feedback the previous pre-decode data for breaking up of the line for second cycle access.

Detailed Description Text (375):
The status bits need to be dual-port to read and write in the same clock cycle. The ICTAGV is organized as two sets of 64 rows by 224 columns and two sets of 64 rows by 128 columns. Each of the first two sets includes seven-bit tag addresses, and each of the last two sets includes three-bit tag addresses and the SU or LV bit.

The two status bits are dual-port RAM cells. The SU uses the delayed PC to write, and the LV bit has the snooping index from L2. The ICTAGV uses 64 rows for dual-port RAM and quick reading of tag addresses. For optimal performance, the layout of the columns should be 32 RAM cells, precharge, 32 RAM cells, write buffer and senseamp. The row decoder should be in the middle of the array to drive 112 or 96 columns each way, and the precharge and the row decoder should cross in the middle of the array. The row decoder for the dual port RAM should be located at one end of the array. The self-time column is used to generate internal clock for each set of the array. The precharge is gated by the ICLK signal. The status bits multiplexed by the TAGHIT signal should be gated by the ICLK signal to be valid for the second ICLK. The above layout is to ensure the minimum routing for the TAGHIT signal.

Detailed Description Text (405):
The global branch prediction method is an independent branch predictor, not a part of the Icache. FIG. 21 is a block diagram of the global branch predictor. Of the many different types of global branch prediction, processor 500 uses the global branch prediction which has the highest ratio of correct predictions. The prediction entries are indexed by an exclusive OR of the PC and the branch shift register. This global branch prediction has a correct prediction of 89.24% based on tsim; the prediction improves as more branch history bits are used in the prediction. A single shift register records the branches taken and not taken by the most recent n conditional branches. Since the branch history includes all branches, global branch prediction takes advantage of two types of patterns: 1) the direction taken by the current branch may depend strongly on the other recent branches, and 2) duplicating the behavior of local branch prediction (patterns of branches in loops). To match the number of entries in the Icache, the global branch prediction has 2048 entries with two targets per entry. It is organized with 256 rows of 8-way associative storage. Eighth bits are needed to index the branch prediction table. The PC uses bits 11:4 for indexing the branch prediction table.

Detailed Description Text (434):
The ICNXTBLK is organized as 5 sets of 64 rows by 256 columns, 1 set of 64 rows by 196 columns, 1 set of 64 rows by 96 dual-ported columns, and 1 set of 64 rows by 64 dual-ported columns. Each of the first two sets consists of 2.times.4 bits of successor index, the next two sets consists of 2.times.4 bits of successor index and 2.times.4 bits of the byte position, the next two sets consists of 2.times.2 bits bimodal counter, 2.times.2 bits predictor counter, and 2.times.3 bits 8-way associative, and the last two sets consist of the 3 bits way-prediction and two bits target selection which are dual-ported RAM cells. The least significant bits of the counters are dual-ported and updated on every cycle. To minimize routing and implementation of the branch holding register, the same associated bits of the two branch targets should be laid out in two sets opposite each other. The branch successor index is selected by the way and target prediction to access the ICACHE in next clock cycle. Because of this speed path in way prediction for reading the Icache in the next cycle, the array for ICNXTBLK is 64 rows instead of 128 rows as for the ICSTORE array. For optimal performance the layout of the column should be 32 RAM cells, precharge, 32 RAM cells, write buffer and senseamp. The row decoder should be in the middle of the array to drive 96 or 112 column each way, and the precharge and the row decoder should cross in the middle of the array. The self-time column is used to generate internal clock for each set of the array. Precharge is gated by ICLK. The ICNXTBLK has two different outputs; the first output in the first cycle is based on the way-prediction and the second output in the second cycle is based on TAGHIT. If the two outputs do not select the same set, or are not both not taken, the reading of instruction in the second cycle will be invalidated, creating a bubble in the pipeline. The second output should be gated with TAGHIT and ICLK to be valid in the second cycle. The way-prediction which uses the return stack may create a speedpath, depending on where the return stack is implemented.

Detailed Description Text (648):
Since the processor clock cycle is reduced to 4.5 ns, reading of the cache takes an

entire clock cycle to get data. The clock is single phase, and the array needs to generate its own self-time clock. The self-time clock uses the same cache column self-time line. As the line is precharged to a high level, the precharge is disabled and the array access is enabled. As the line is discharged, the row driver and senseamp are disabled. The precharge takes 1.7 ns and the current timing for TAGHIT from the self-time clock with 64 rows is 2.8 ns for a total time of 4.5 ns from rising edge of ICLK. The reading of data occurs 2.0 ns from the self-time clock with 64 rows or 0.8 ns before the rising edge of ICLK. The ICSTORE can be implemented using larger arrays, 128 rows by 256 columns. The reading of instructions would take all of 4.5 ns ICLK in this case. All other arrays, ICTAGV, ICPRED, and ICNXTBLK, are 64 rows. The align logic in the ICPDAT takes 6-7 gates, the shifting of X86 instruction bytes to the decode unit can be done by the middle of the second ICLK. The fast path instructions should allow the decode units at least 2.5 ns in the second ICLK for calculation of the linear address.

Detailed Description Text (716):
IDxLOCK--Output to indicates the lock prefix is set for this instruction for serialization.

Detailed Description Text (741):
INSLSxB(5:0)--Output from decode units indicates the prefix values. bit 5--data size, bit 4--address size, bit 3--lock, bit 2:0--segment registers.

Detailed Description Text (849):
IDxLOCK--Output to indicates the lock prefix is set for this instruction for serialization.

Detailed Description Text (850):
INSLSxB(5:0)--Output from decode units indicates the prefix values. bit 5--data size, bit 4--address size, bit 3--lock, bit 2:0--segment registers.

Detailed Description Text (907):
MIDPREF(5:0)--Output from MIU prefix decode to decode units indicates the prefix values. bit 5--data size, bit 4--address size, bit 3--lock, bit 2:0--segment registers. This can be from the same prefix decoding as the 2-cycle access.

Detailed Description Text (909):
The decode units, in most cases, can decode instructions, generate operand addresses, and dispatch to the functional units individually. There are a few exceptions where global controls are needed. In a few cases, the line of instruction has to be dispatched in a sequence over many clock cycles. Examples include MROM instructions, SIB-byte instructions, two-cycle fast path instructions, and conditional branch instructions which are taken. In these cases the lines of instructions are modified and refreshed instead of accepting a new line of instruction. Partial line dispatching should be detected in the second ICLK. Other conditions to halt the line of instructions before dispatching to the functional units in the next ICLK are the reservation full, the LOROB full, and the Load/Store buffer full. These halt conditions will stop the pipeline in the decoder from advancing.

Detailed Description Text (911):
Each decode unit detects the conditions for breaking up the line. The two-cycle fast path and MROM instructions are indicated by the functional-byte. SIB-byte instructions are detected by two functional bits not being set between the opcode byte and the displacement/immediate byte. The taken branch instruction is from information from the ICNXTBLK or fast decoding of unconditional branch instruction. The information is sent to the global control to modify and refresh the line of instructions. Some instructions will be changed to NOOP before dispatching to functional units.

Detailed Description Text (922):
FIG. 34 is a block diagram of how two-cycle fast path instructions are handled. The number of prefix bytes included in fast path instructions is limited to three. Allowed prefixes include 0.times.F0 for lock, 0x66 for toggling between 16 or 32 bit data, 0x67 for toggling between 16 or 32 bit address, 0x0F for two-byte opcode, and six more prefixes for segment register override. The prefix bytes are indicated by the number of functional bits set beginning with the start-byte. The decoding of fast path instructions allows only one prefix. In cases where instructions have more than one prefix bytes, an extra cycle is needed to shift the instruction and decode the prefixes. The number of bytes shifted is based on the number of functional bits set beginning with the start-byte. The prefixes combine with the MODRM to provide the size information to the stack cache and register file. The decoding of the prefixes are done before the next cycle begins.

Detailed Description Text (946):
IDPREF(5:0)--Output from 2-cycle prefix decode to decode units indicates the prefix values. bit 5--data size, bit 4--address size, bit 3--lock, bit 2:0--segment registers.

Detailed Description Text (962):
FIG. 35 is a block diagram of the layout of the processor 500 instruction decode unit. The Idecode includes six decode units. Decode units 0 and 6 are modified to accommodate the wrapping of instructions from one cache line to the next. The global blocks are: MROM interface unit, the prefix decoding and control for 2-cycle fast-path instructions, the return stack and controls for branch instructions, and global decoding controls. The MROM interface unit includes global registers accessible by MROM instructions.

Detailed Description Text (1038):
INSLSxB(5:0)--Input from decode units indicates the prefix values. bit 5--data size, bit 4--address size, bit 3 lock, bit 2:0--segment registers.

Detailed Description Text (1044):
EXCHGSYNC--Output indicates exchange instruction resynchronization to Icache. This occurs when an exchange with a masked underflow is retired. It is a special resynchronize exchange with alternate entry point.

Detailed Description Text (1082):
WBEXCHG--Output to register file indicates the exchange instruction being retired. It causes the permanent remapping register to be updated from the write-back bus.

Detailed Description Text (1109):
For internal exceptions from the functional units, LSSEC, and SRB, the exception entry in the LOROB will be retired in order. As with branch mis-predictions, the pipe and fetching should stop on an exception indication. When all entries before the exception entry are completed and retired, the exception procedure is initiated. All entries in the LOROB, the functional units, and LSSEC will be purged. The exception routine will be fetched. The LOROB is responsible for generating the entry point into the MROM exception routine or new PC into the Icache. No state is updated when a trap is taken. The processor simply fetches from an appropriate entry point and allows the microcode to perform the necessary state modifications. It is up to the microcode to save the current EIP on the stack before the user*s trap handler is called.

Detailed Description Text (1122):
INSLSxB(5:0)--Input from decode units indicates the prefix values. bit 5--data size, bit 4--address size, bit 3--lock, bit 2:0--segment registers.

Detailed Description Text (1138):
Due to out-of-order execution, the ADD instruction may read the old data at addr1

from the stack cache. The STORE instruction will modify the data at addr1 during execution. For correct operation, the ADD instruction should be restarted. The above problem is preferred to as SC.sub.-- read-after-DC.sub.-- write dependency. Another problem is DC.sub.-- read-after-SC.sub.-- write dependency, the DC.sub.-- read is an indirect addressing. This dependency is illustrated in the following example:

Detailed Description Text (1228):
On all external interrupts, the entry point is generated locally by the LOROB at the time the redirection is initiated. If the maskable interrupt is level sensitive while the NMI is edge sensitive. FLUSH and INIT are also treated as edge sensitive asynchronous interrupts, similar to NMI. The NMI is taken, it cannot be taken again before an IRET is executed. The microcode maintains a series of global flags that are inspected and modified by many of the trap handler entry points, and the IRET instruction. It is also the responsibility of the microcode to detect the NMI and delay the NMI until after executing of the IRET, the MROM allows only one level of NMI. Many other aspects of nested trap control (double fault, shutdown, etc.) will be handled with this microcode mechanism. There is no hardware support for any of this. When an enabled trap condition arises, the LOROB takes it at the next available window.

Detailed Description Text (1265):
EXCHGSYNC--Output indicates exchange instruction resynchronization to Icache. This occurs when an exchange with a masked underflow is retired. It is a special resynchronize exchange with alternate entry point.

Detailed Description Text (1359):
INSLSxB(5:0)--Input from decode units indicates the prefix values. bit 5--data size, bit 4--address size, bit 3 lock, bit 2:0--segment registers.

Detailed Description Text (1368):
ROBLSYNC--LOAD/STORE RESYNC--1-bit--Set when functional units return valid results with resync status. The load/store hits in the Icache for self-modifying code. The next instruction should be re-fetched from the Icache.

Detailed Description Text (1519):
In previous X86 architectures, quick operand accesses were limited to only eight registers or slower accesses to one or two read ports for memory (data cache) operands. The line of code above can access all of its operands out of the register file or out of the stack cache which are both very quick. The current model only uses one push per dispatch position. A speculative copy of ESP is available to the six linear address adders. These adders can quickly (1/2 cycle; end of ICLK2) determine base pointer and stack pointer relative linear addresses which use 32 bit displacements. ICLK3 is used to determine multiple pushes, ESP/EBP add, or subtract updates (i.e., SUB ESP,0.times.20). Three pushes are allowed per line. A MOV EBP, ESP and a POP EBP instruction will update the speculative copy of EBP during the 3rd ICLK. Aligned 32 bit accesses to the stack cache are done quickly while unaligned accesses that cross 32 bit boundaries are converted to DC accesses. Unaligned reads are done by the LSSEC as two separate reads and two consecutive cycles. Unaligned writes from the LOROB to the stack cache also take two cycles since there is only one writeback port per position. A DC write to the stack cache only sets a "w" but does not perform the actual write. During ICLK4 the ESP and EBP relative accesses (either read or write) are done on the stack cache. A write access will set the "w" bit for the LOROB line (one being dispatched) on the corresponding stack cache line. For example, if the 2nd LOROB line is being dispatched, a write to stack cache line 1, way 0 would set the 2nd "w" bit on stack cache line 1, way 0. Any line with a "w" bit set cannot be replaced (sent back to the data cache if modified) until the writeback and clearing of the "w" bit. Accesses which read from the stack cache in the 4th ICLK do not set any bits but only read the appropriate data and send it to the operand steering unit.

Detailed Description Text (1528):
FIG. 43 is a block diagram of the look-ahead ESP and EBP register models. Base pointer relative additions (EBP and displacement) occur in the 2nd ICLK for eight bit displacements. The linear address can be used at the beginning of the 4th ICLK. A maximum of 3 pushes are allowed per line. The stack cache linear address requires that Flat segmentation is being used; otherwise an additional cycle would be needed to bring in the segment using the transfer bus and add it to the sum of the base pointer and displacement. SIB addressing that uses the ESP with no index will also require another cycle after the 2nd ICLK. The 3rd ICLK pipe stage is dedicated to setting up for multiple Pushes and for moving the ESP (subtracting and updating ESP). If a "MOV EBP,ESP", "ADD EBP, imm", "SUB EBP, imm", "POP EBP", "MOV ESP, EBP", "ADD ESP, imm", or "SUB ESP, imm" is detected, the linear address calculation stage will attempt to update a speculative copy of EBP or ESP (3rd ICLK) and continue issuing subsequent opcodes that use base pointer relative addressing. When an opcode that modifies EBP or ESP, does not use the previous encodings, the subsequent opcodes will be stalled in the pipeline until EBP/ESP is non-speculative.

Detailed Description Text (1532):
The operand is a memory location which is either Locked or Non-Cacheable

Detailed Description Text (1533):
When a Locked access occurs to a stack cache or data cache line, processor 500 will first drive a bogus Locked Read on the external pins in order to maintain control of the bus. The load can occur from the stack cache or data cache. Then if the line is modified, the stack cache or data cache line must be copied back to external memory. Finally, the Locked single cycle write will occur externally following by Unlock.

Detailed Description Text (1682):
FIG. 49 is a block diagram of the bus structure for the reservation stations. Each reservation station has a front latch which triggers on the rising ICLK edge and a back latch which triggers off a self timing delay after the front latch. A MUX before the front latch allows either new data to come in from the higher numbered reservation station (or operand steering section for RS2) for from the back latch of the same reservation station. The information received from the back latch of the same reservation station could of course be different than the information that the front latch originally sent. For example the front latch may receive a tag for both its A and B operands and then send this information to the back latch. The back latch could receive the forwarded data for the A operand, reset the VAT (valid A tag) bit, and send this information back to the front latch or to the next front latch. The reservation stations shift their information to the next lower numbered reservation station only when new information is coming in. Next the front latch would send the information to the back latch, and the tag comparators might detect a match for the B tag. The back latch would latch in the B operand and send it onto the FNCU input MUX. The RSCTL maintains the juggling act of which operations end up in which reservation stations. An operation will always stay in its current reservation station unless it is shifted to the next or sent to the FNCU for evaluation.

Detailed Description Text (1684):
FIG. 50 is a reservation station timing diagram. Right after the front latch fires, a self timing circuit begins a timing delay before the back latch can latch in its data. The tags for an FNCU operation are sent out towards the end of the previous cycle; these are latched in at the beginning of the current cycle along with the new reservation station information. Then the tag comparisons begin to take place. The new information along with tag comparison matches from all three reservation stations goes to the RSCTL unit to begin the process of deciding which operation gets sent to the FNCU next cycle, which back latches need to receive forwarding

operands, and how the front end MUXes will be set up for juggling operations around the reservation stations at the beginning of next cycle.

Detailed Description Text (1689):
3. Linear address--base pointer or stack pointer linear address does not hit in the LOROB or in the stack cache; or is not allowed to hit in the stack cache (extremely rare: LOCKed) and does not hit in the LOROB.

Detailed Description Text (1969):
3. Linear address--base pointer or stack pointer linear address does not hit in the LOROB or in the stack cache; or is not allowed to hit in the stack cache (extremely rare: LOCKed) and does not hit in the LOROB.

Detailed Description Text (2030):
The load store section can perform single-cycle accesses of two memory based operands. It can also perform out-of-order loads requested by the functional units. The stores always go in order and are performed as pure writes, rather than read-modify-writes. The data cache is a linear cache, dual ported for the two concurrent accesses, 16/32 KB 8-way set associative with way prediction.

Detailed Description Text (2195):
D is the dirty bit that indicates that the line has been previously modified. This information is used during a store when the TLB is accessed to determine whether the corresponding dirty bit in the page table entry is correctly set. If the dirty bit in the page table entry is not set then an exception must occur to write the dirty bit in the external page table entries so that the page gets written back to external memory.

Detailed Description Text (2202):
The data array is effective dual ported due to interleaving. Each bank will be physically laid out as two 64 rows.times.256 column arrays. The speed target for processor 500 does not allow a contiguous array larger than 64 rows. During a clock cycle, at most two banks can be accessed. The banks are selected based on the bank select bits 4:2 of the port addresses. The data array is byte addressable via the use of byte enables. Two sets of byte enables are generated per clock corresponding to the two banks being accessed. The byte enables are generated using the operand size information as well as bits 1:0 of the linear address. The byte enables are critical to doing stores in a single cycle for aligned accesses since stores are done as pure writes instead of the read-modify-writes. Unaligned accesses and 8/16 bit accesses use byte enable information in the same fashion as well.

Detailed Description Text (2211):
Stores are accomplished as pure writes and not read-modify-writes. The dcache supports byte write capability which allows pure writes. The byte enables used to do this are generated based on the operand size and bits 1:0 of the port linear address. The dcache will support single cycle accesses for stores if the store is to the predicted way.

Detailed Description Text (2239):
It is further noted that aspects regarding array circuitry may be found in the co-pending, commonly assigned patent application entitled "High Performance Ram Array Circuit Employing Self-Time Clock Generator for Enabling Array Access" filed concurrently herewith by Tran. The disclosure of this patent application is incorporated herein by reference in its entirety.

Detailed Description Text (2242):
Numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. It is intended that the following claims be interpreted to embrace all such variations and modifications.

CLAIMS:

7. The lookahead register value generator as recited in claim 5 wherein said each of said plurality of decode units is configured to detect instructions which modify one of a plurality of registers for which said lookahead control unit stores a corresponding said lookahead register value, and wherein said each of said plurality of decode units is configured to convey a third control value on said plurality of first control buses, and wherein said third control value is indicative of said modification.